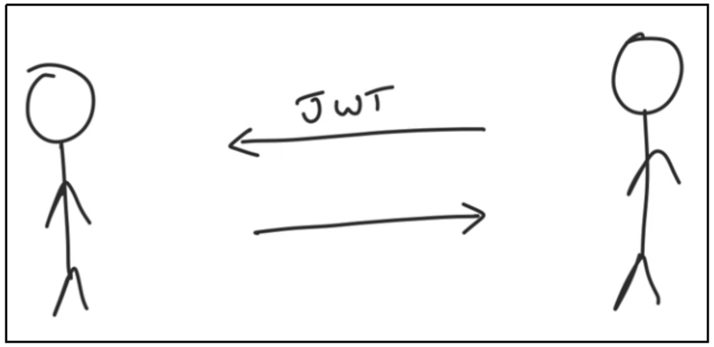
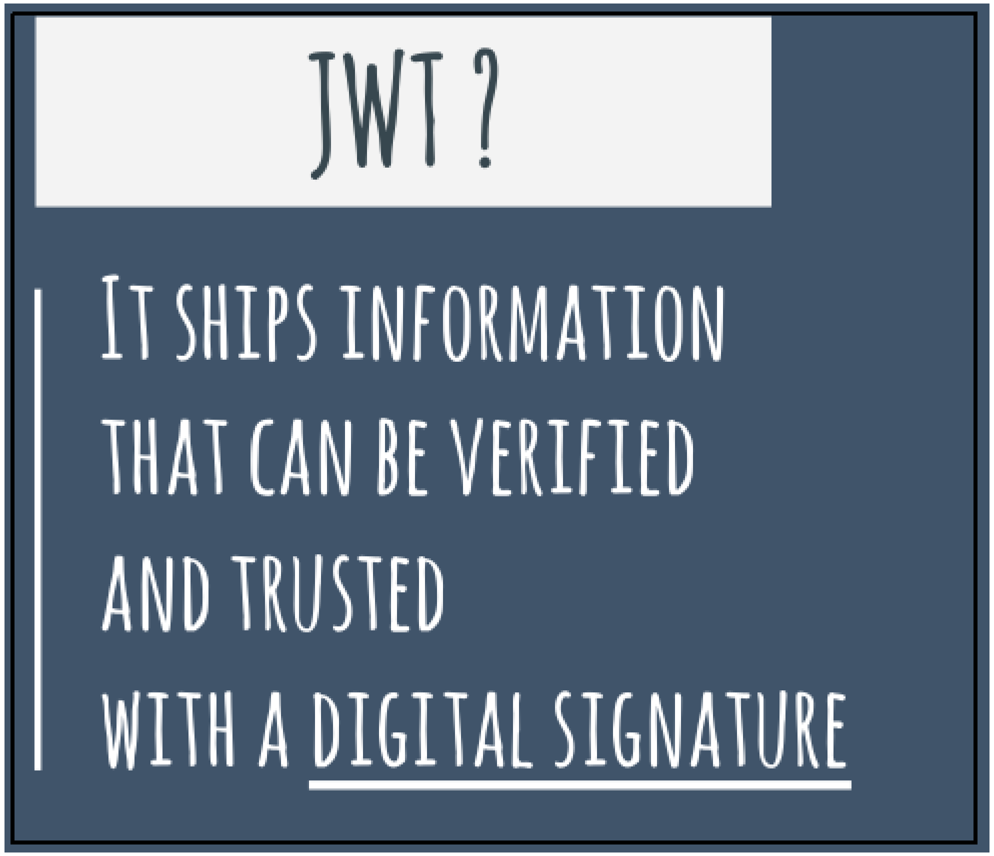
JWT based security

rgupta.mtech@gmail.com



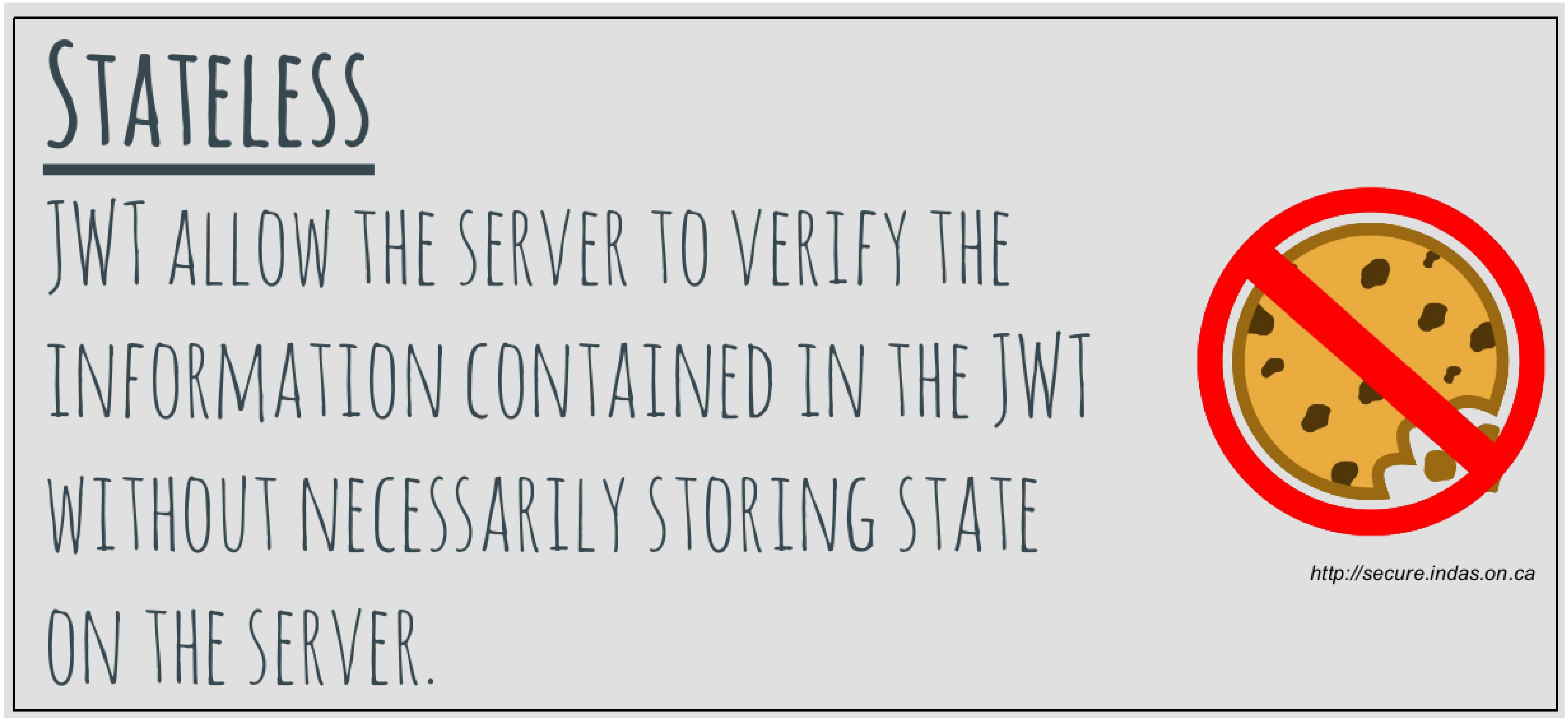
What is JWT?



JWT ?

It ships information
that can be verified
and trusted
with a digital signature

Why JWT?

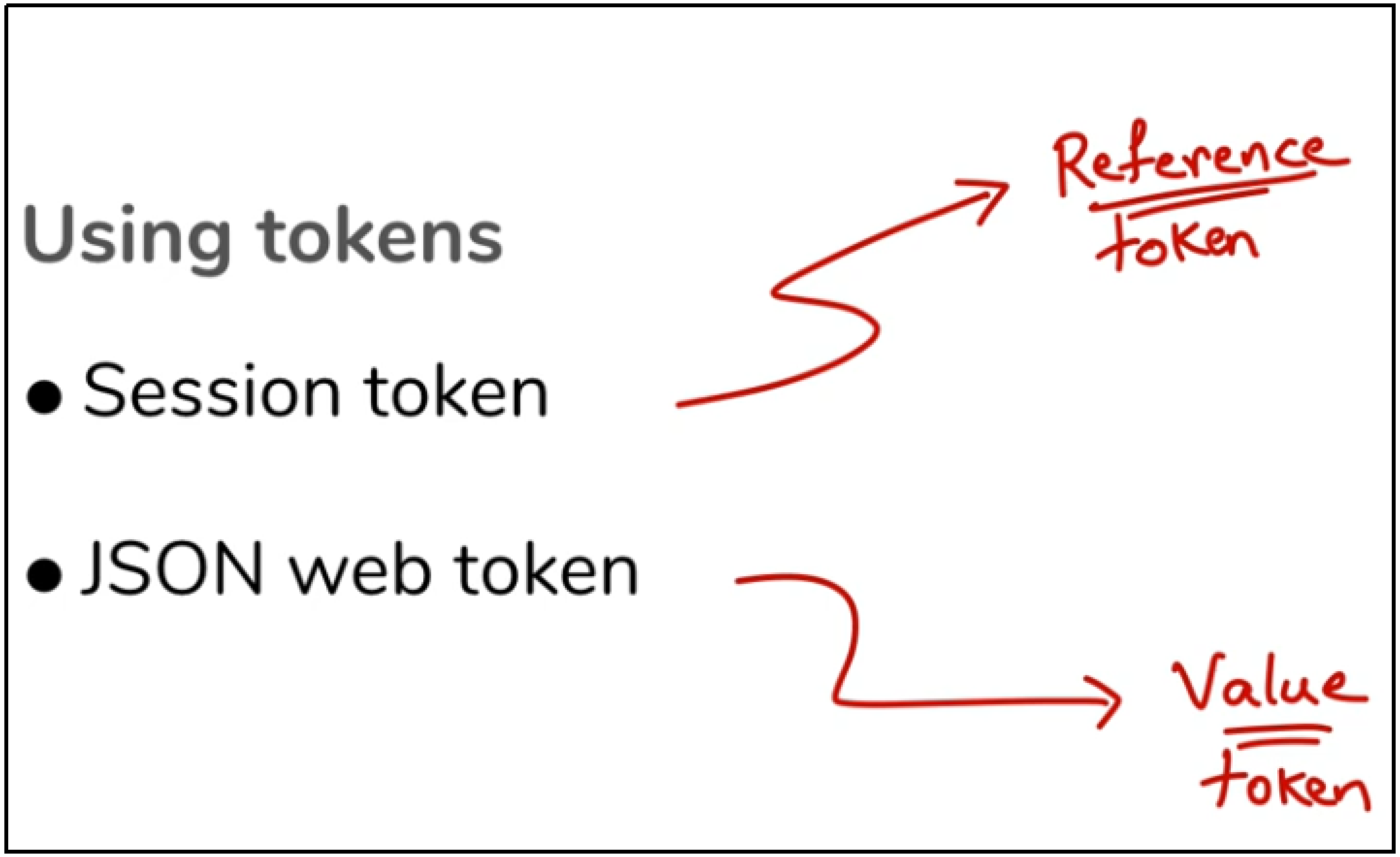


STATELESS

JWT ALLOW THE SERVER TO VERIFY THE INFORMATION CONTAINED IN THE JWT WITHOUT NECESSARILY STORING STATE ON THE SERVER.

http://secure.indas.on.ca

Authorization strategies



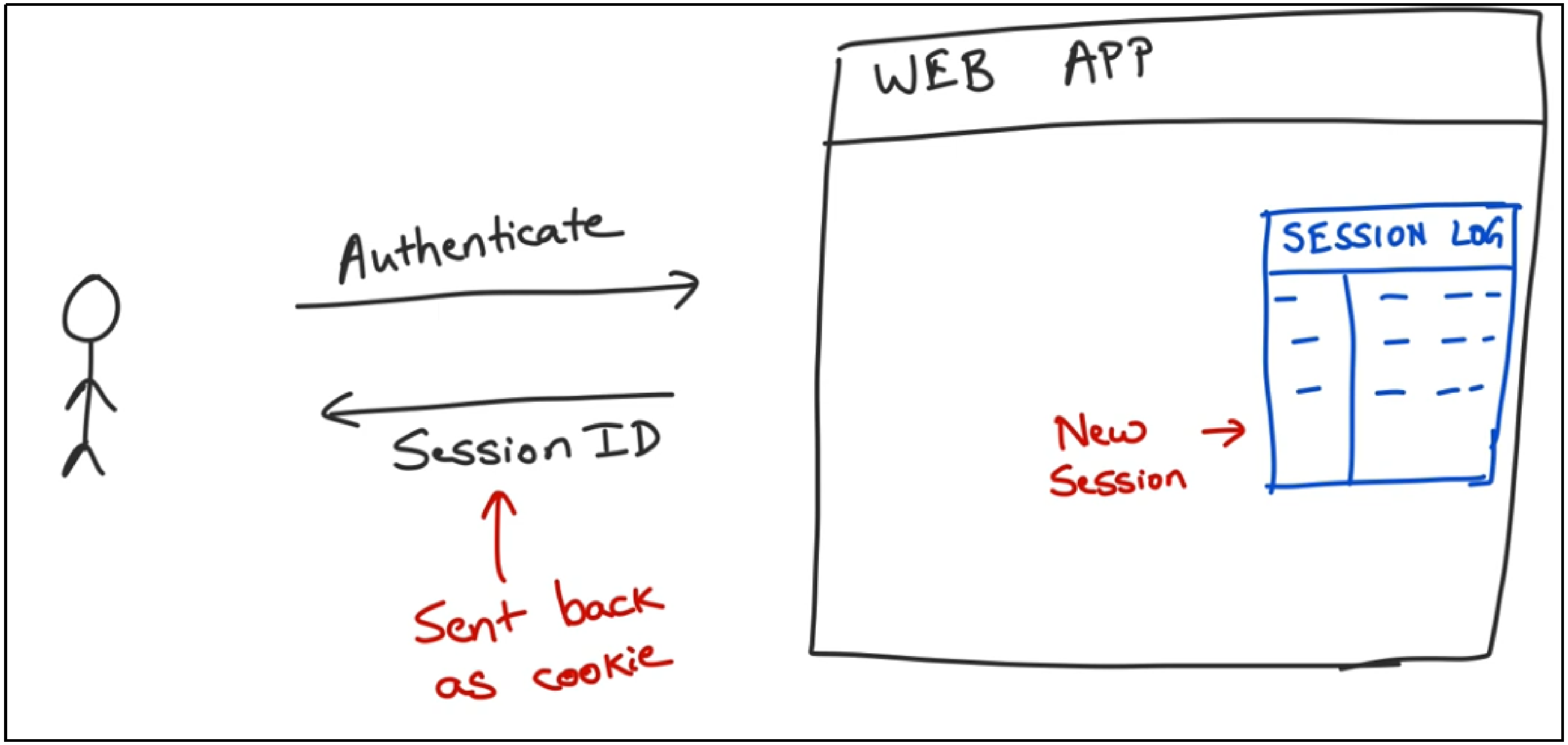
Using tokens

- Session token
- JSON web token

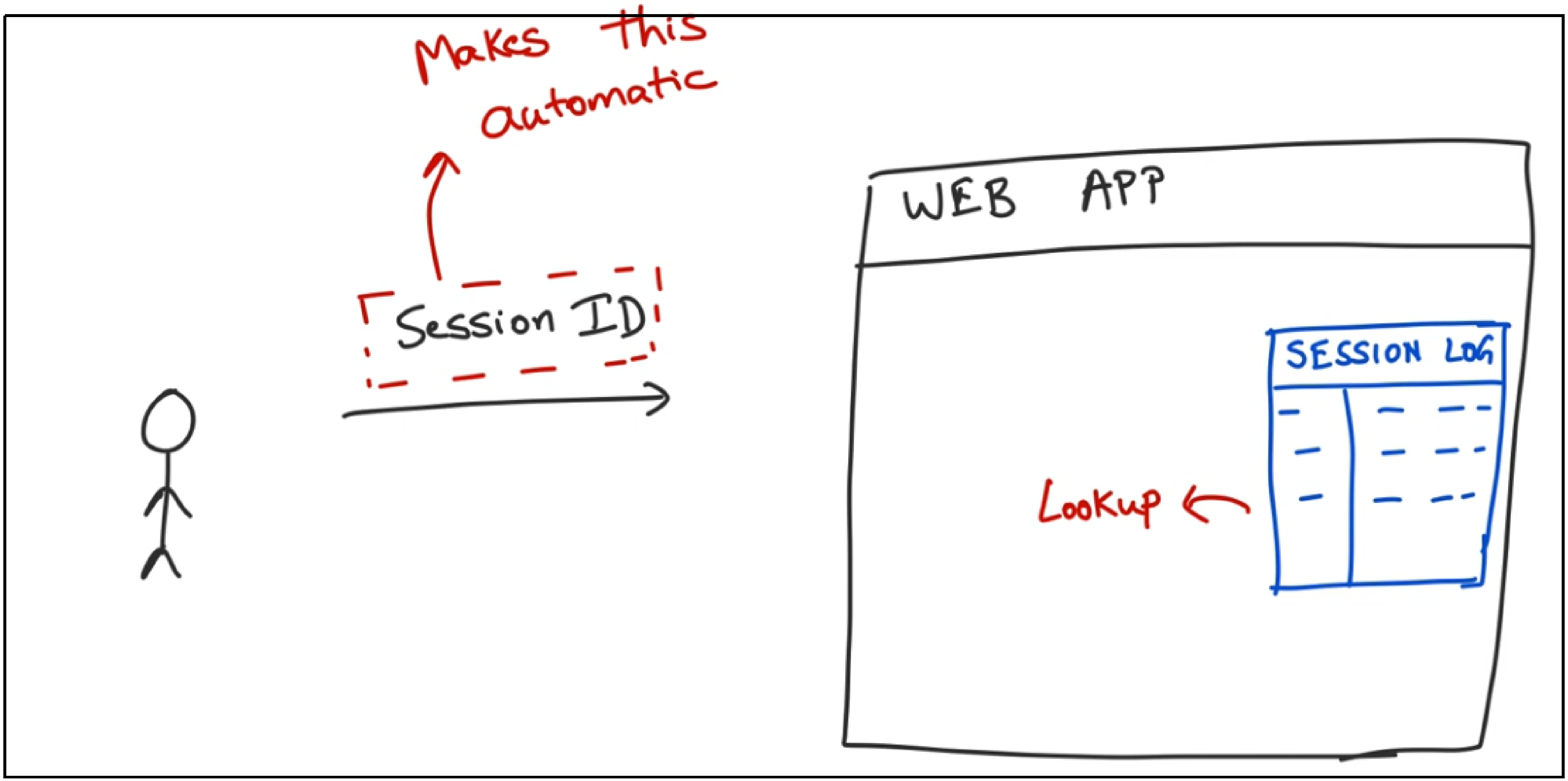
Session ID + Cookies

Most popular mechanism for authorization

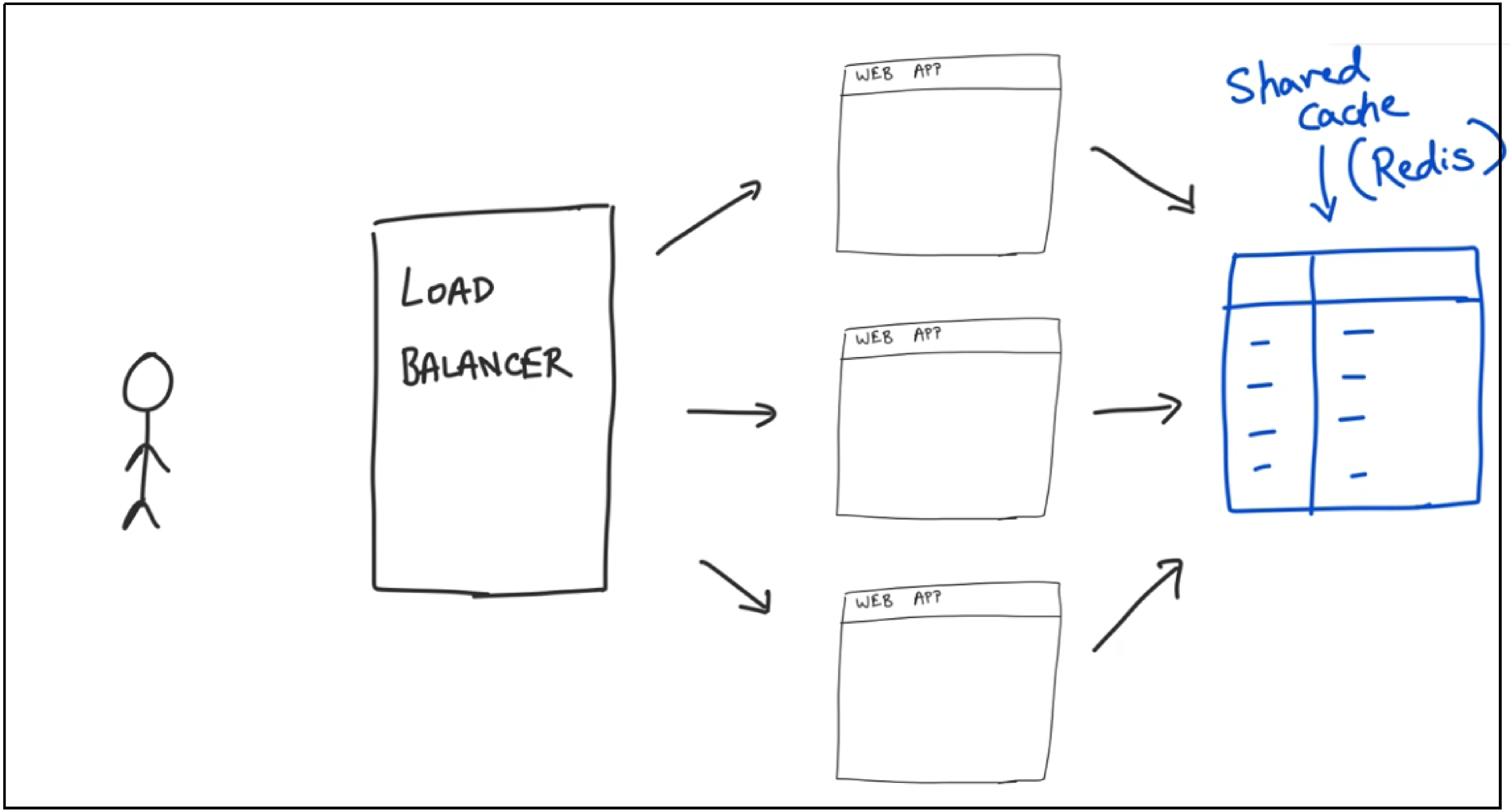
Session based authorization



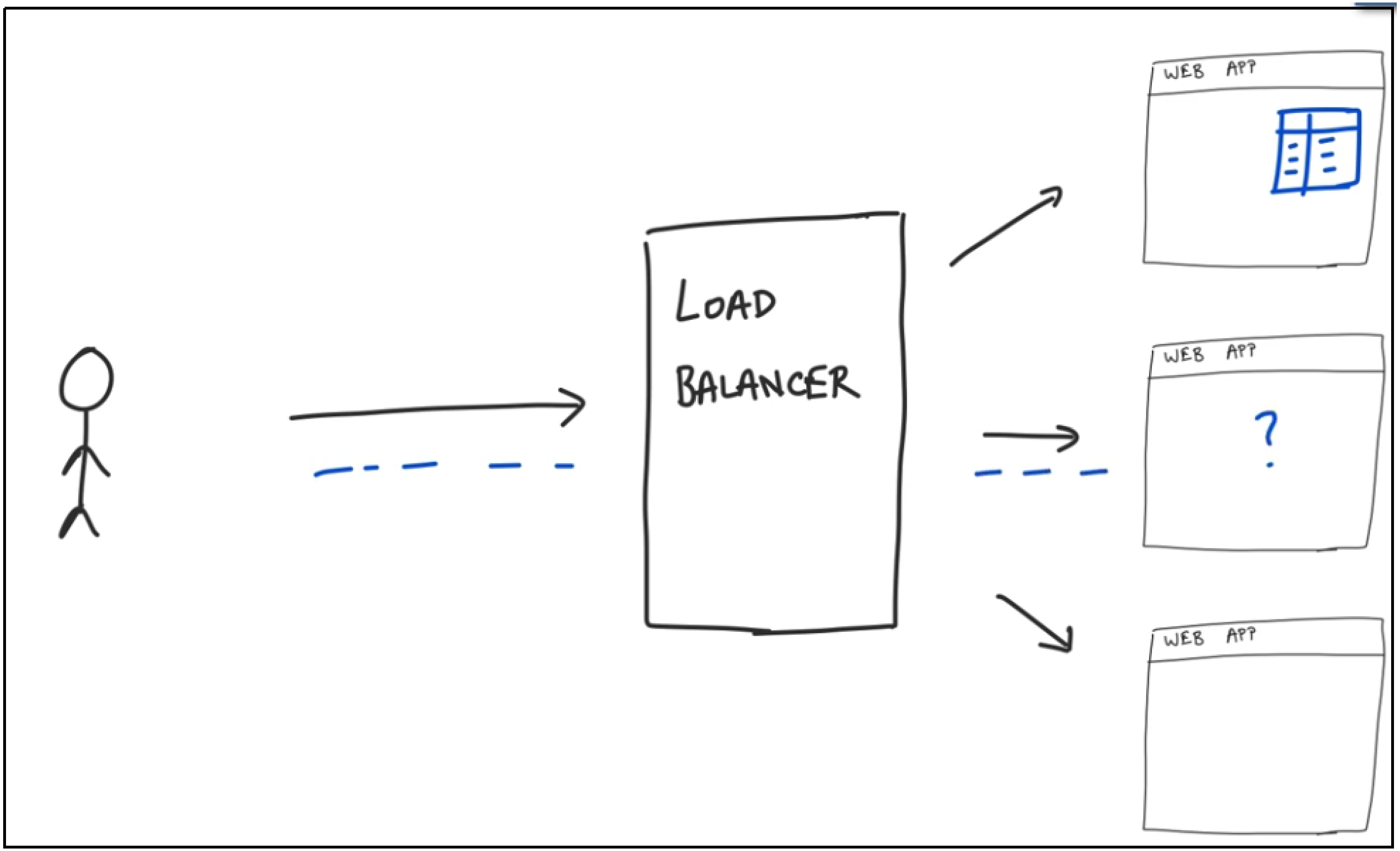
Session based authorization



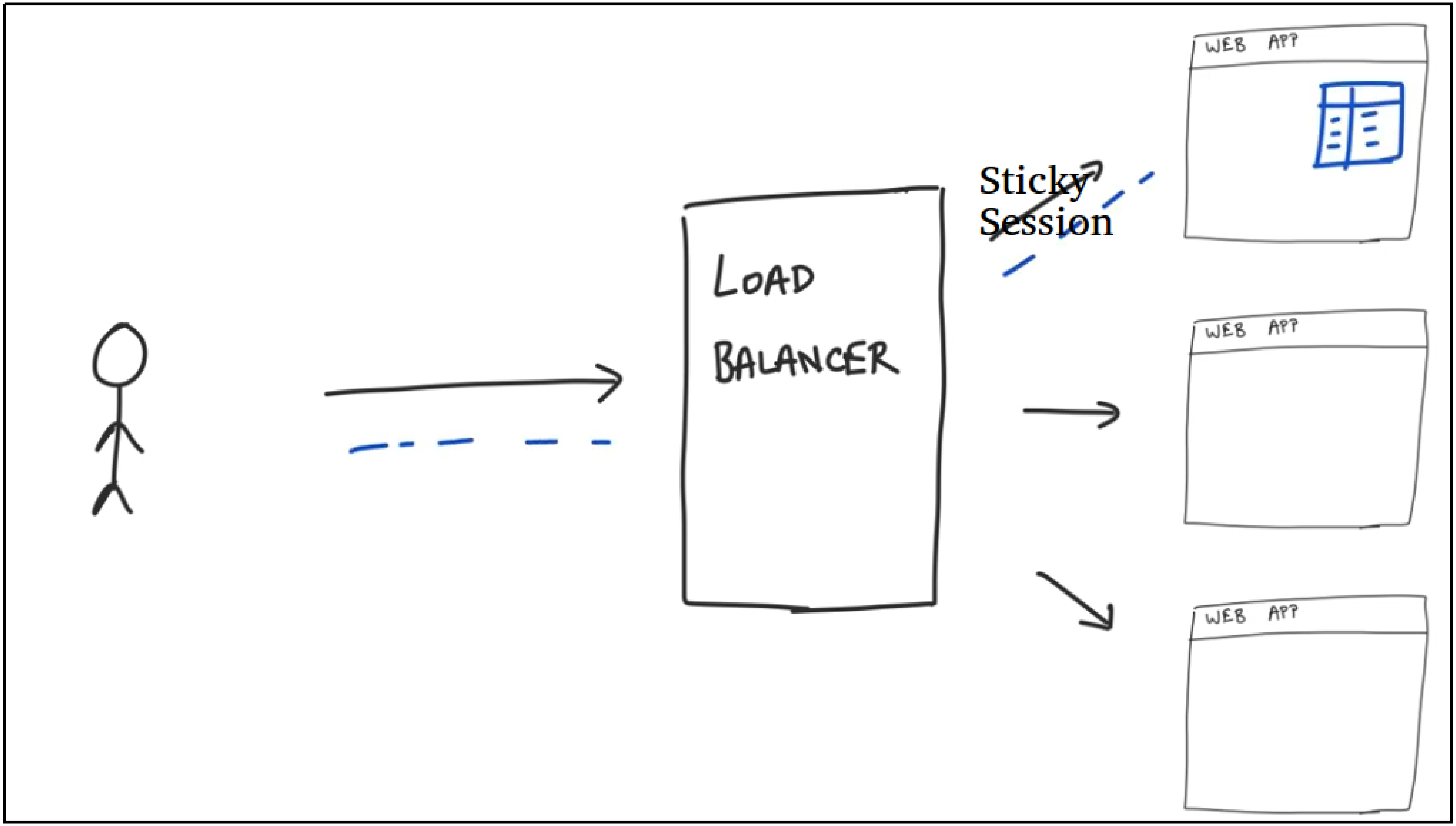
Session based authorization



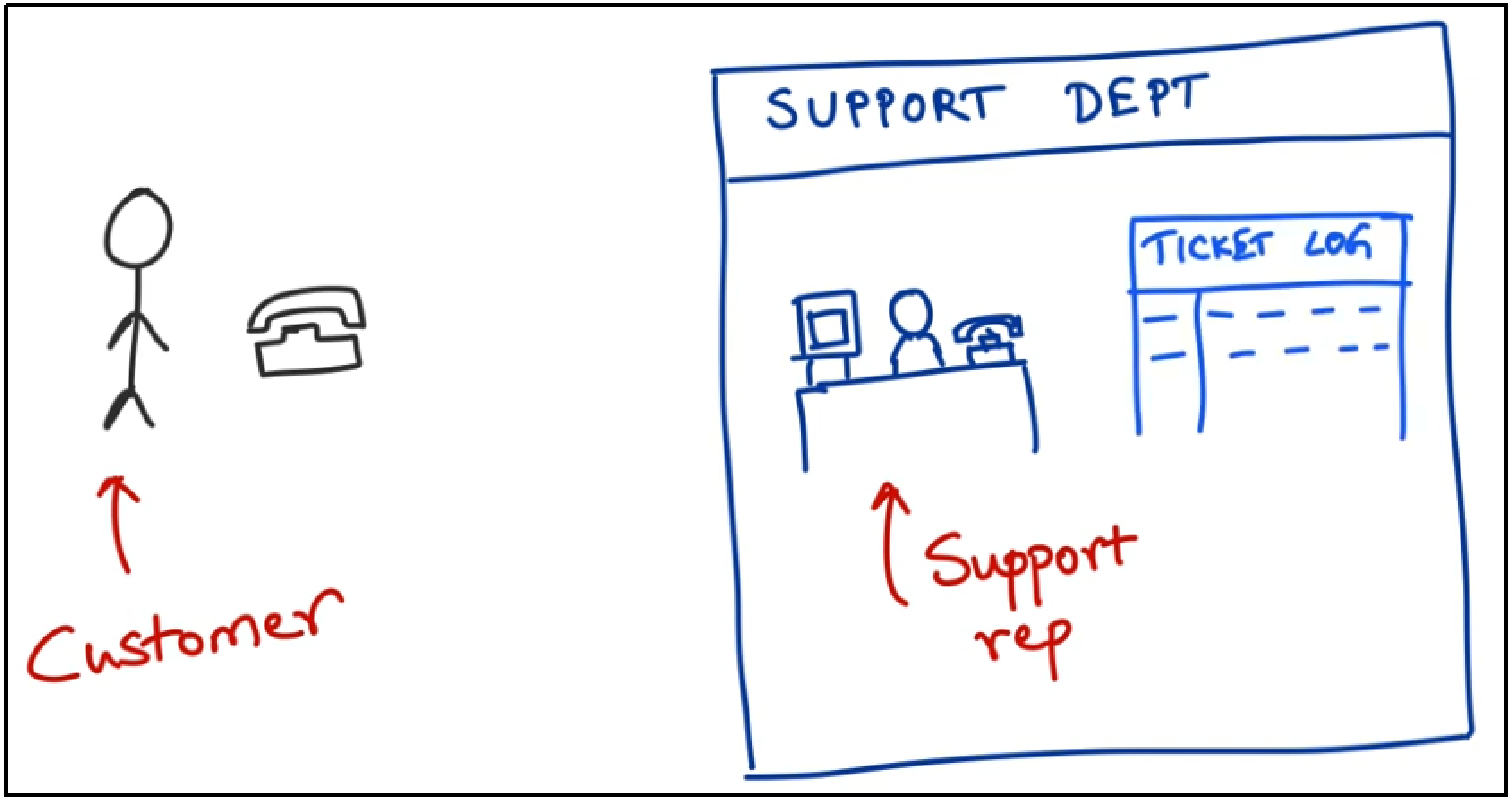
Session based authorization



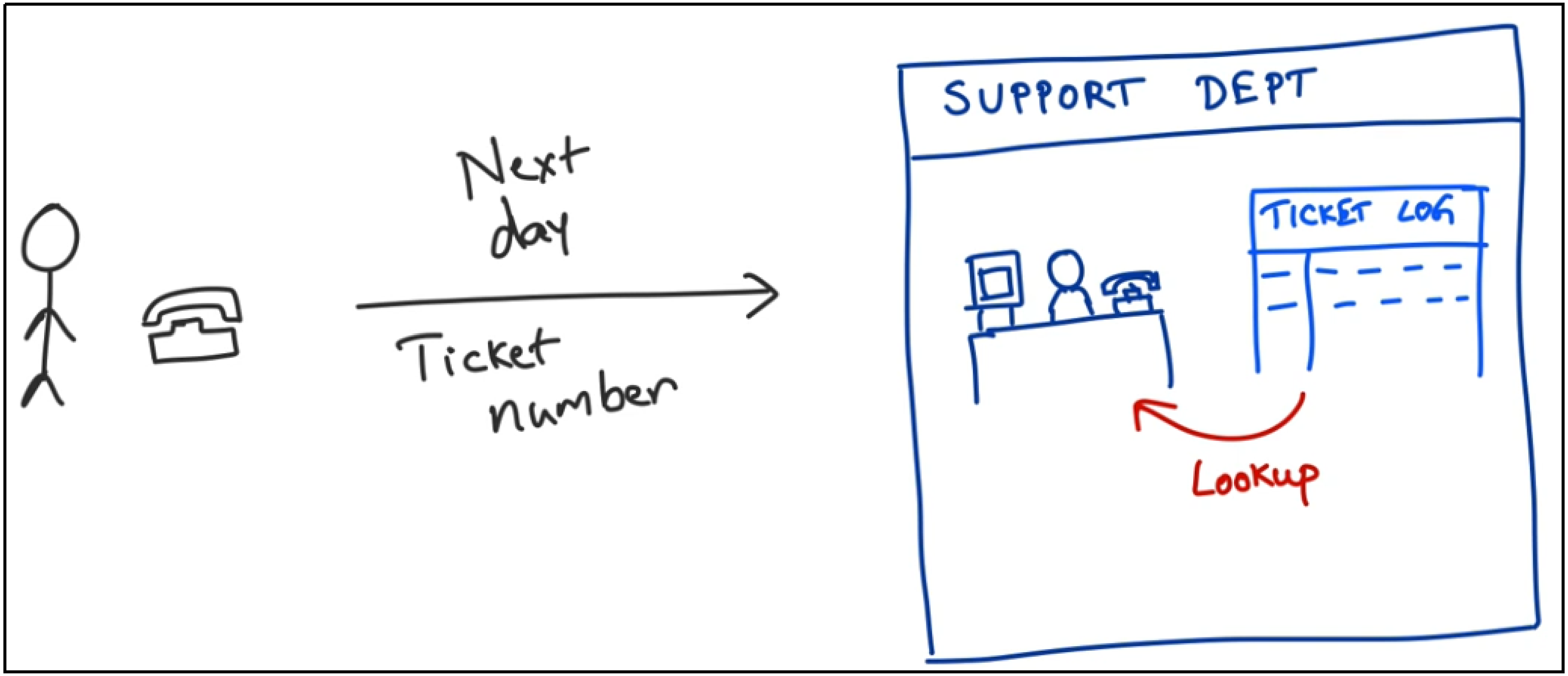
Session based authorization



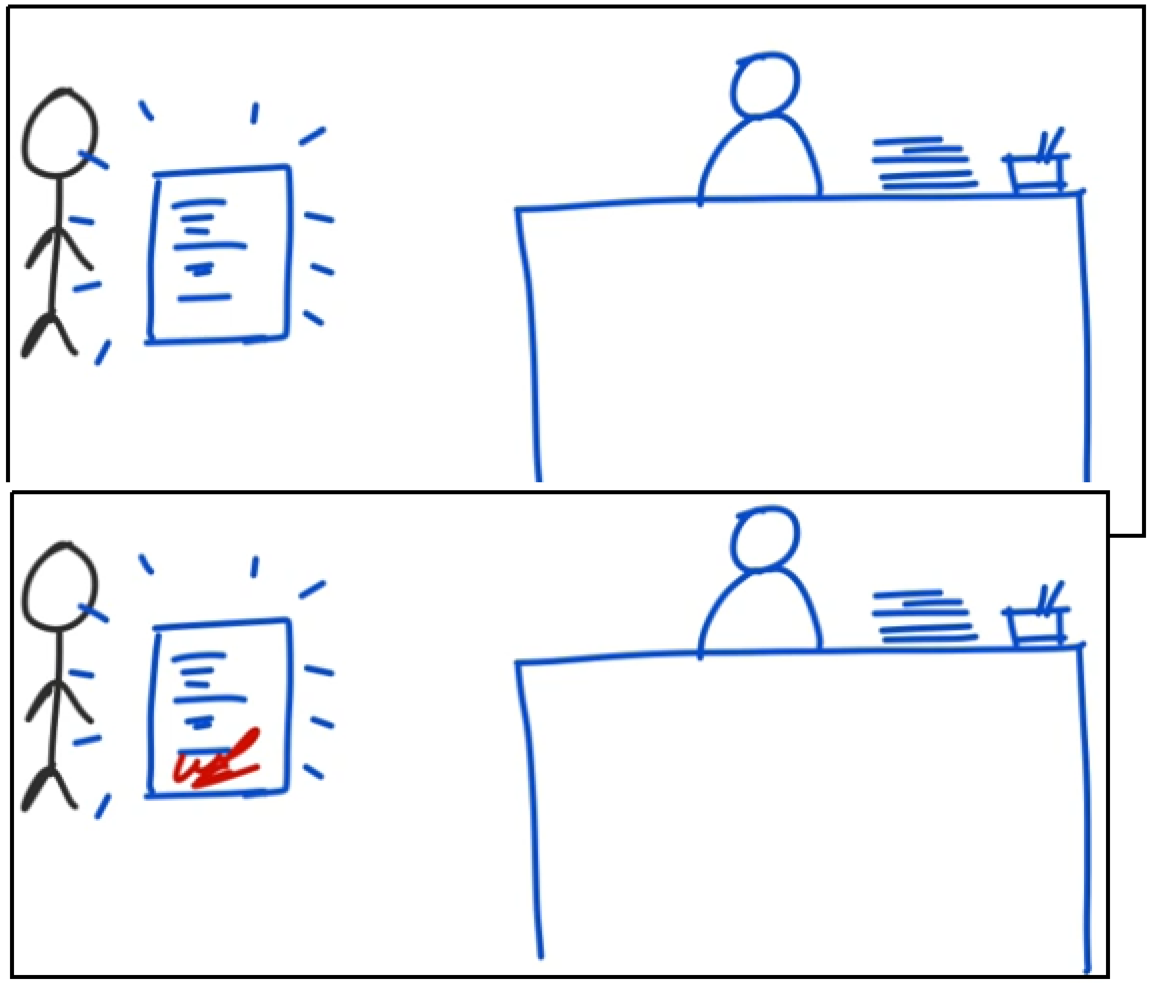
Session based authorization



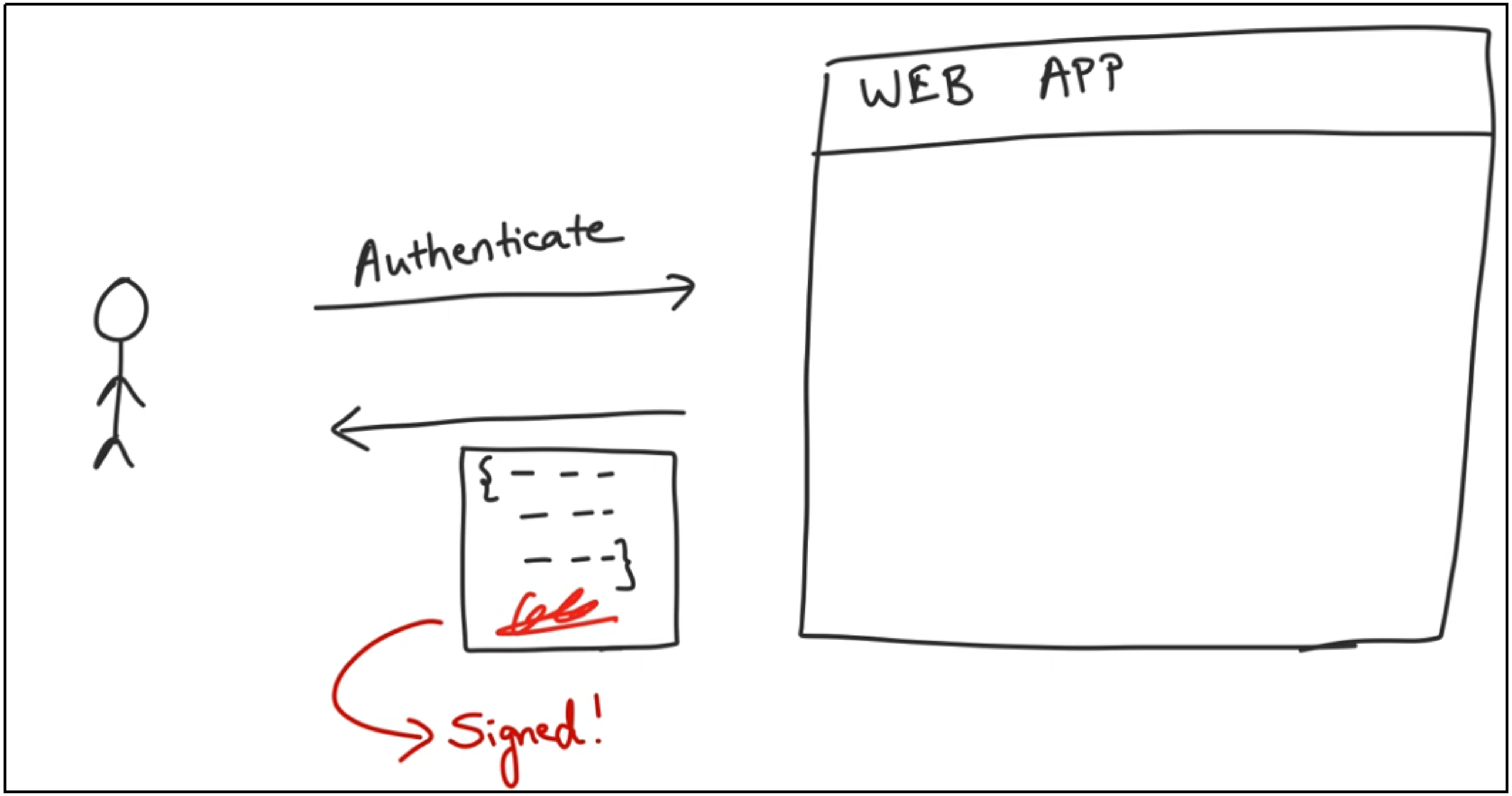
Session based authorization



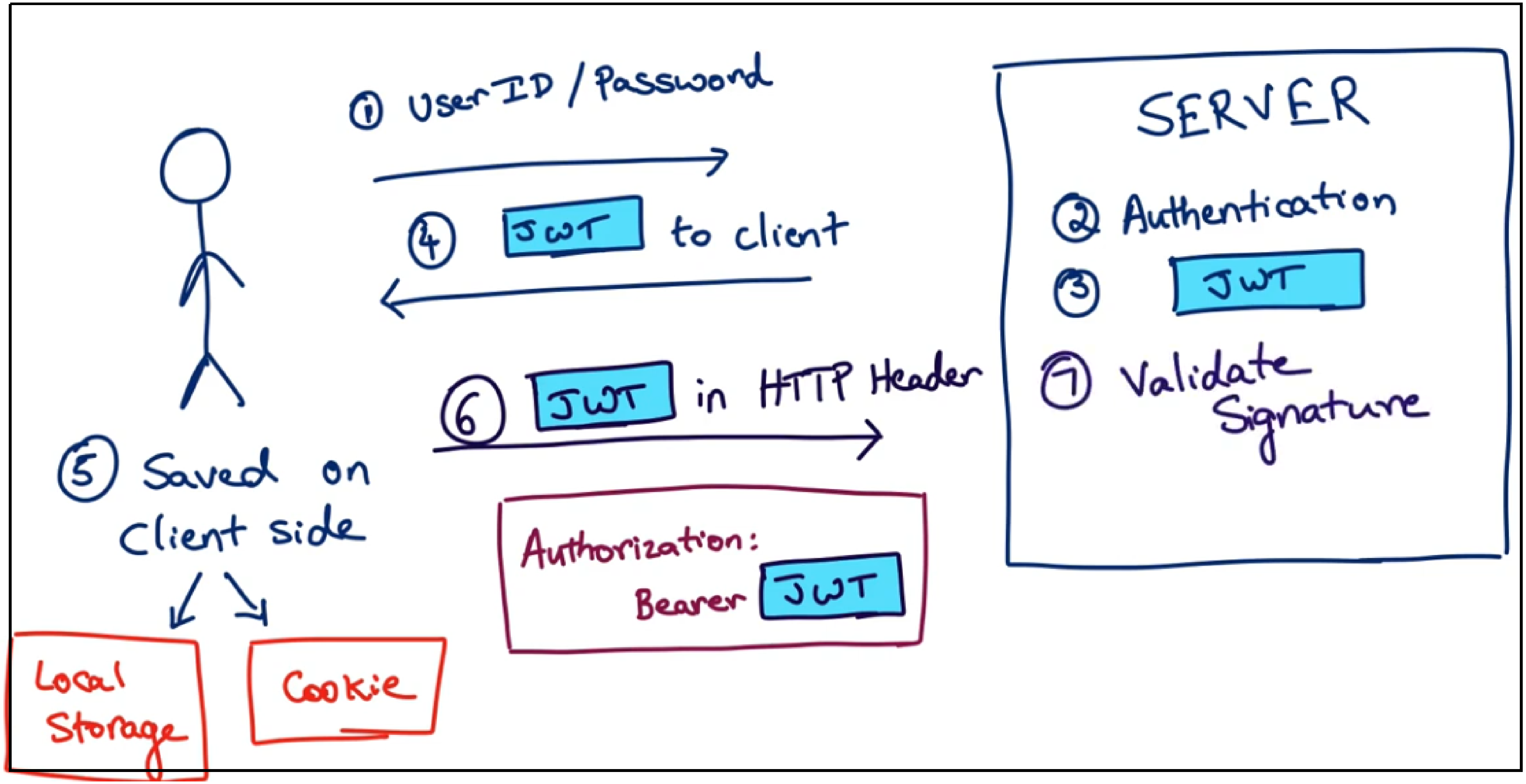
JWT based token



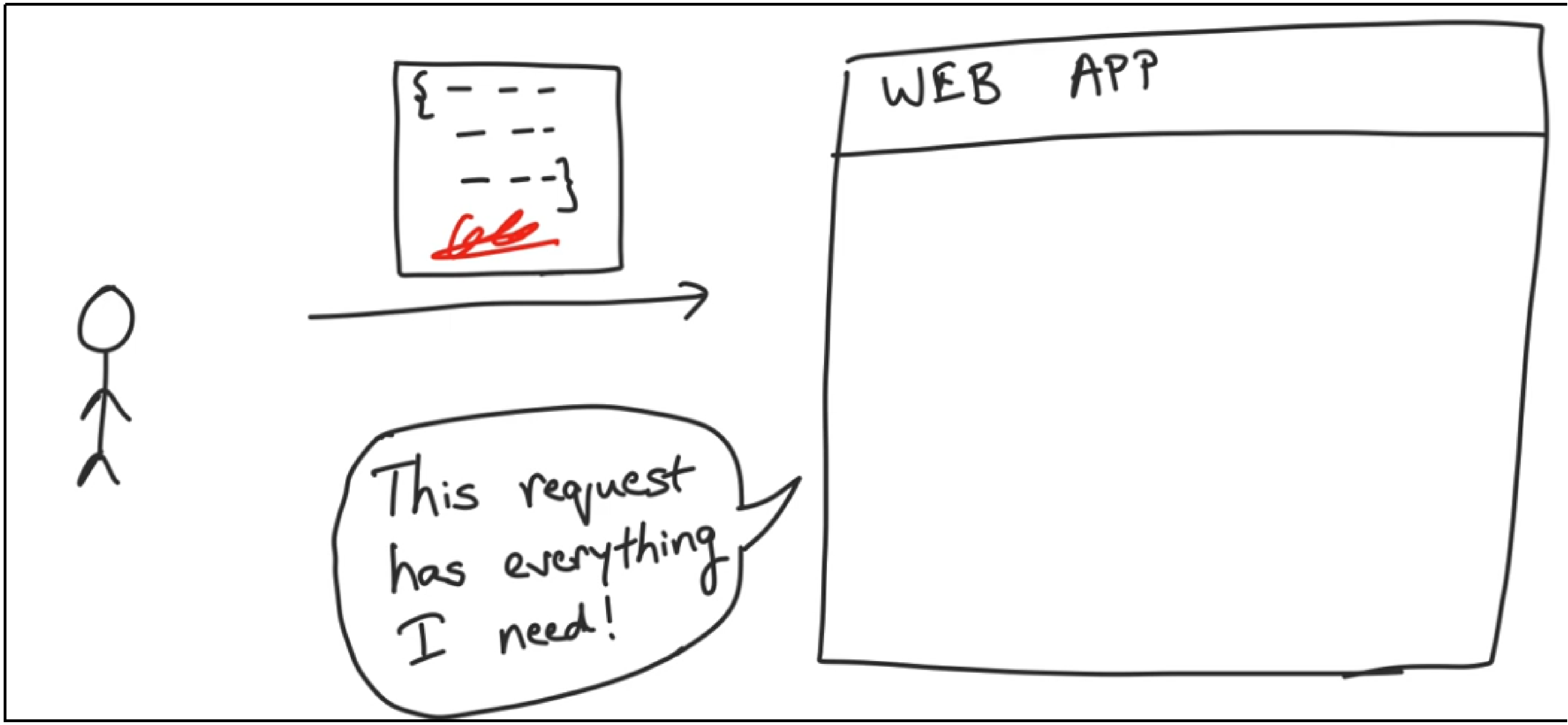
JWT based token



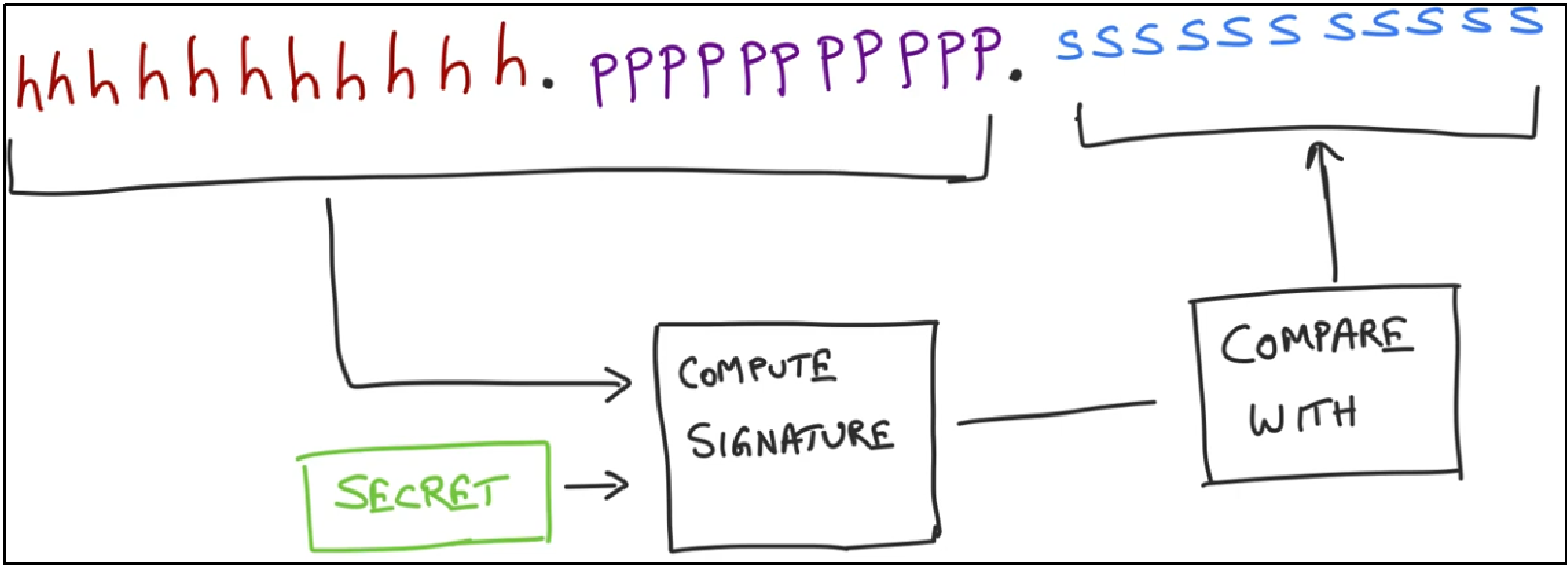
JWT token flow

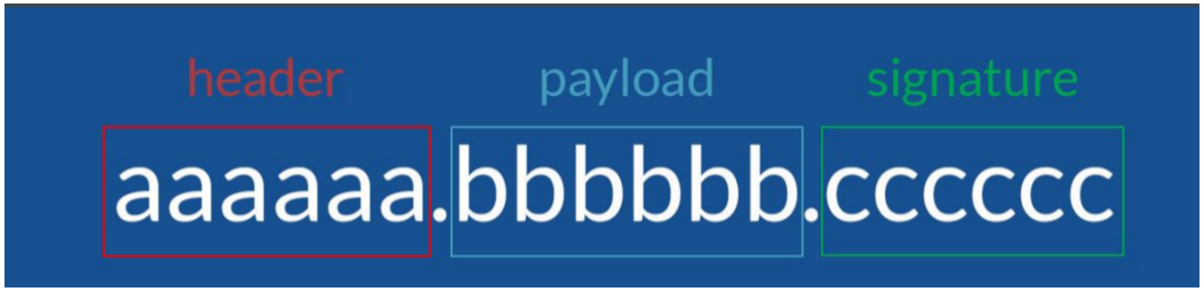


JWT Token flow

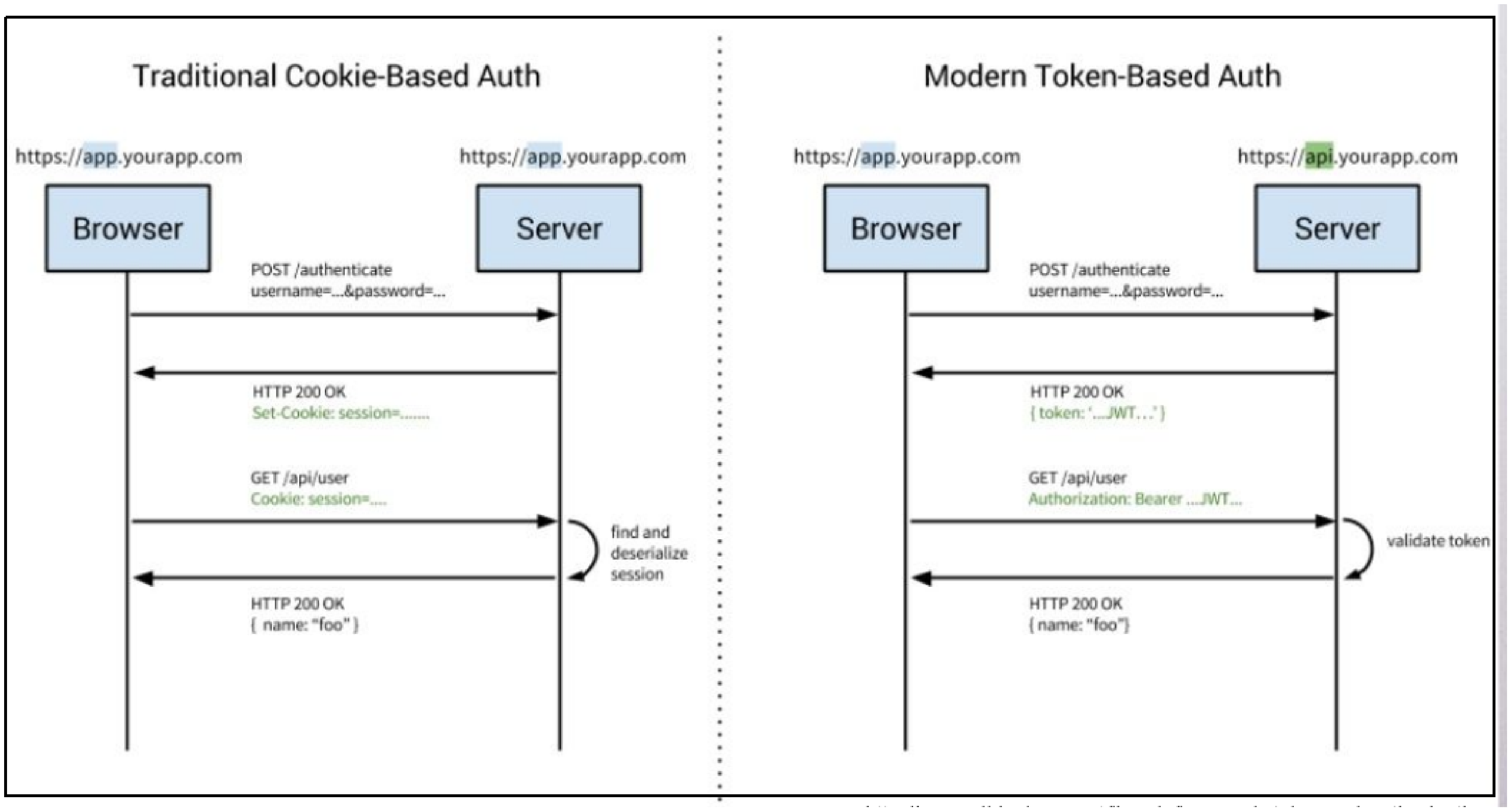


JWT token

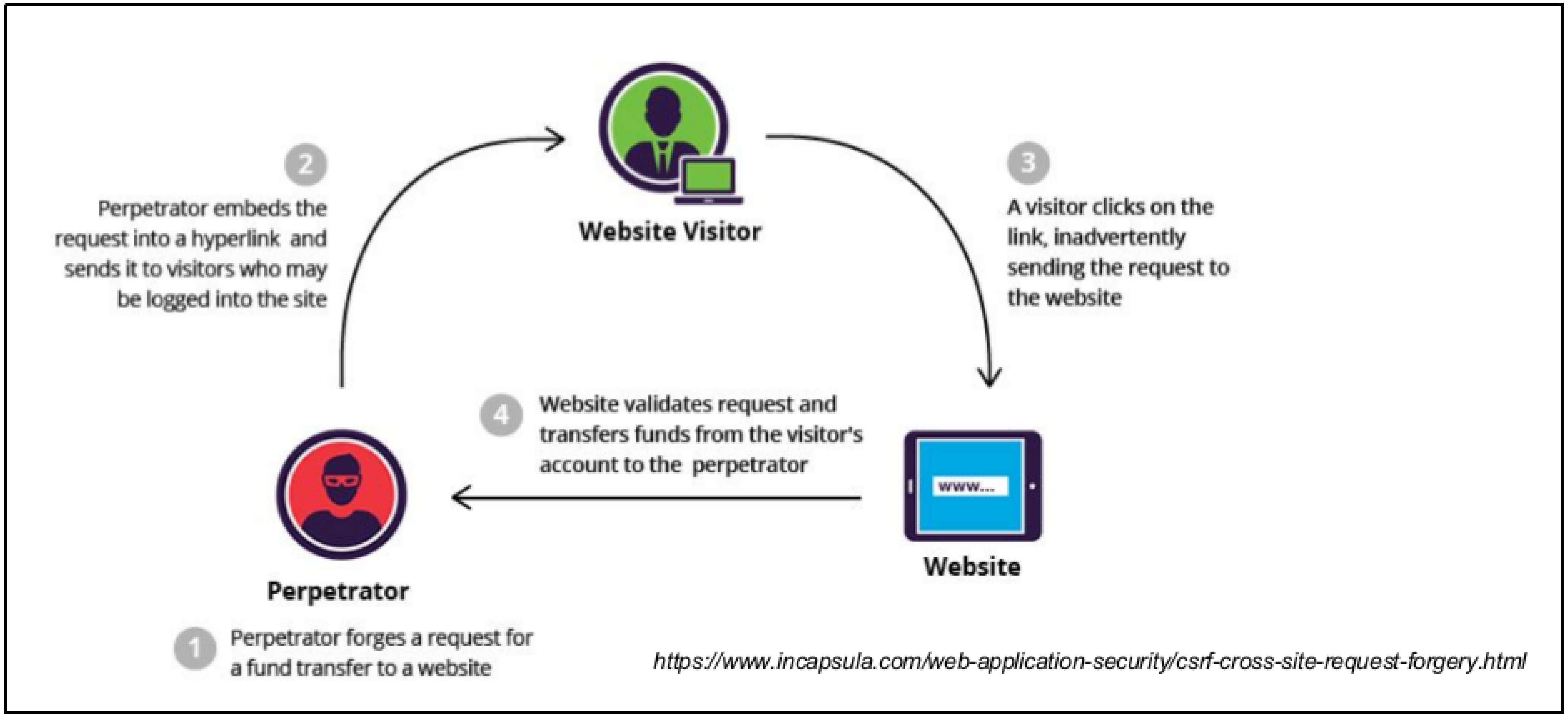

hhhhhhhhhhh.
pppppppppppp.
ssssssssssss
COMPUTE SIGNATURE
SECRET
COMPARE WITH


header
payload
signature
aaaaaa.bbbbbb.cccccc

Cookie vs JWT based authorization



No need to protect against CSRF


Website Visitor
2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1
Perpetrator forges a request for a fund transfer to a website

https://www.incapsula.com/web-application-security/csrf-cross-site-request-forgery.html

Header

Parts of The header :

- Declaring the type, which is JWT
- The hashing algorithm to use

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

Common JWT Signing Algorithms

HS256	HMAC using SHA-256
RS256	RSASSA-PKCS1-v1_5 using SHA-256
ES256	ECDSA using P-256 and SHA-256

Payload

Carry the information that we want to transmit, also called the JWT Claims.

```
{
  "iss": "scotch.io",
  "exp": 1300819380,
  "name": "Chris Sevilleja",
  "admin": true
}
```

Signature

Made up of a hash of the following components:

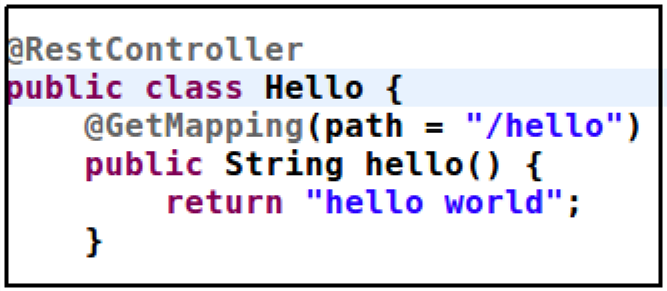
- the header
- the payload
- secret

```
var encodedString =
base64UrlEncode(header) + "." +
base64UrlEncode(payload);


HMACSHA256(encodedString,'secret');
```

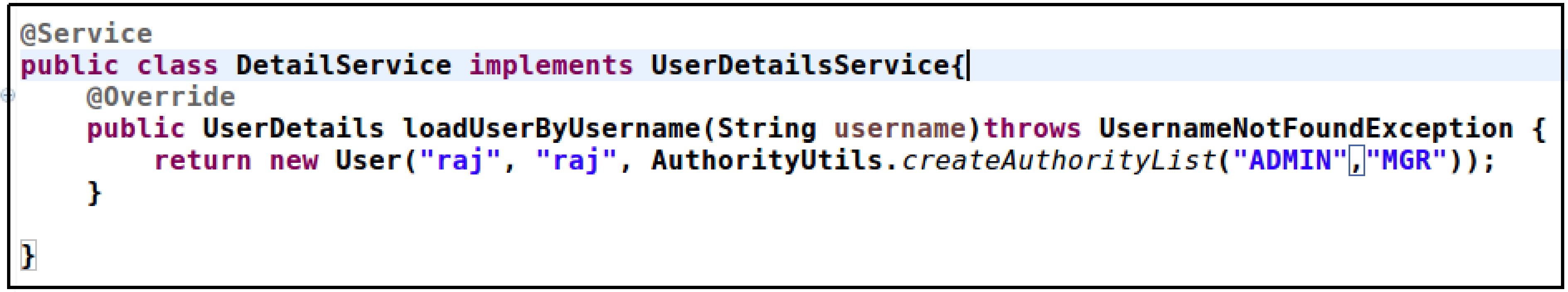
Full JSON of JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.	HEADER
eyJpc3MiOiJzY290Y2guaW8iLCJleHAiOjEz MDA4MTkzODAsIm5hbWUiOiJDaHJpcyB TZXZpbGxlamEiLCJhZG1pbiI6dHJ1ZX0.	CLAIMS
03f329983b86f7d9a9f5fef85305880101d5e302 afafa20154d094b229f757	SIGNATURE

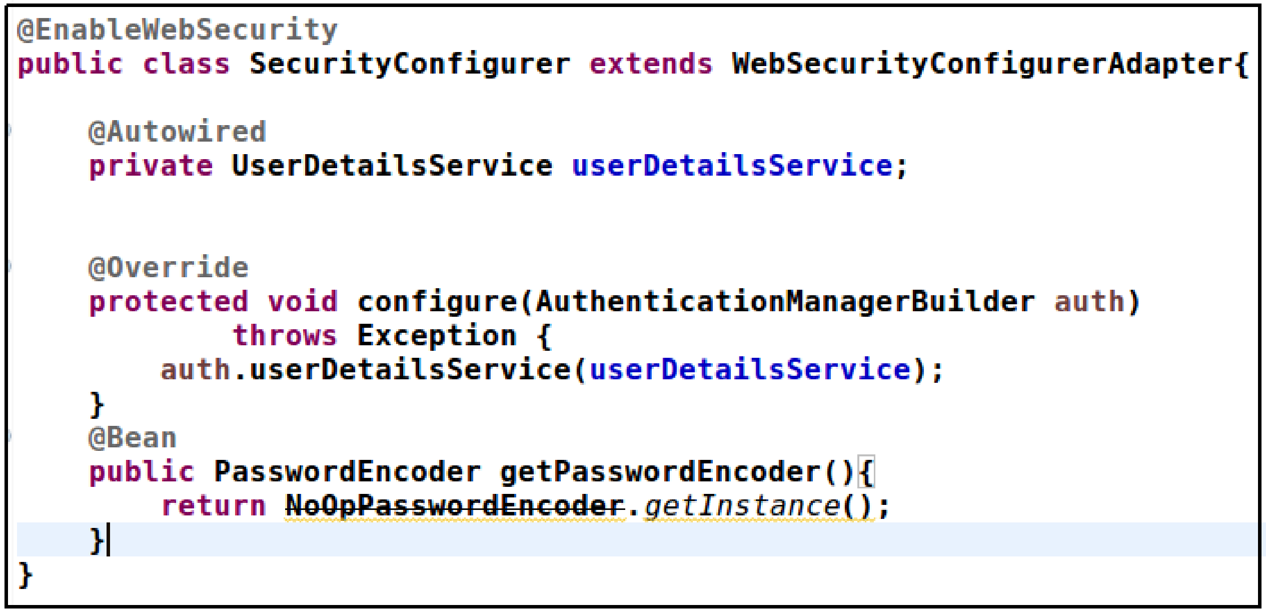
Step1: start with hello world spring boot security configuration



```java
@RestController
public class Hello {
    @GetMapping(path = "/hello")
    public String hello() {
        return "hello world";
    }
```

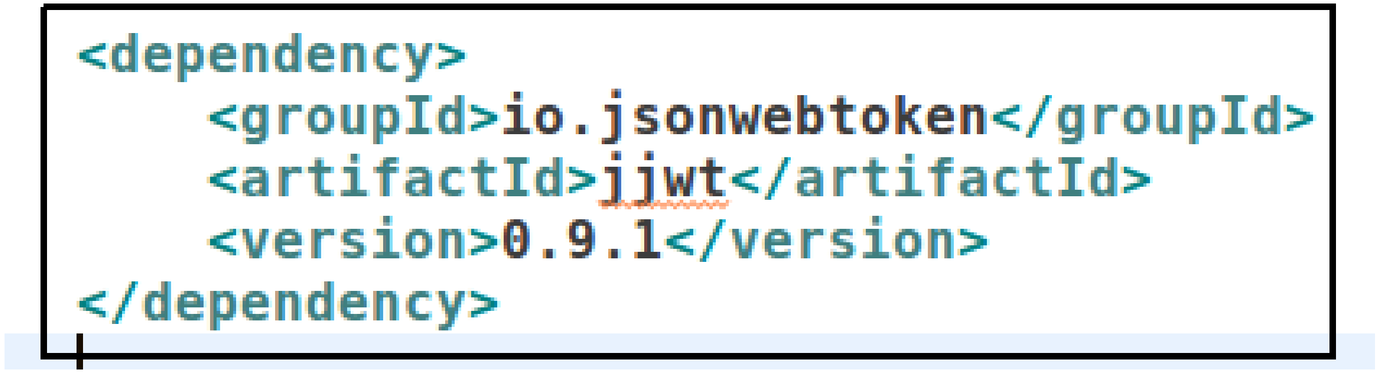


```java
@Service
public class DetailService implements UserDetailsService{
    @Override
    public UserDetails loadUserByUsername(String username)throws UsernameNotFoundException {
        return new User("raj", "raj", AuthorityUtils.createAuthorityList("ADMIN","MGR"));
    }

}
```

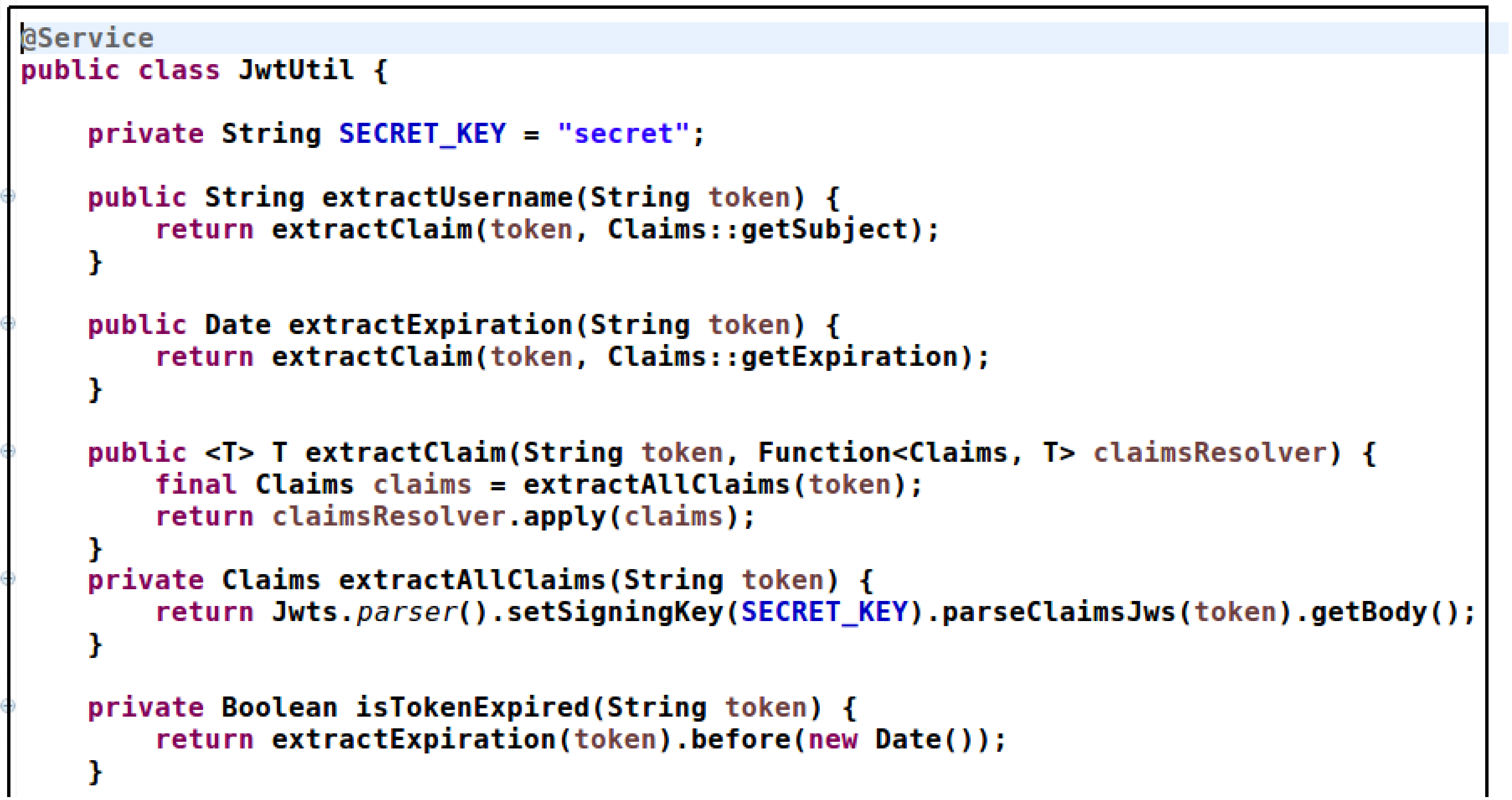


```java
@EnableWebSecurity
public class SecurityConfigurer extends WebSecurityConfigurerAdapter{

    @Autowired
    private UserDetailsService userDetailsService;


    @Override
    protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {
        auth.userDetailsService(userDetailsService);
    }
    @Bean
    public PasswordEncoder getPasswordEncoder(){
        return NoOpPasswordEncoder.getInstance();
    }
}
```

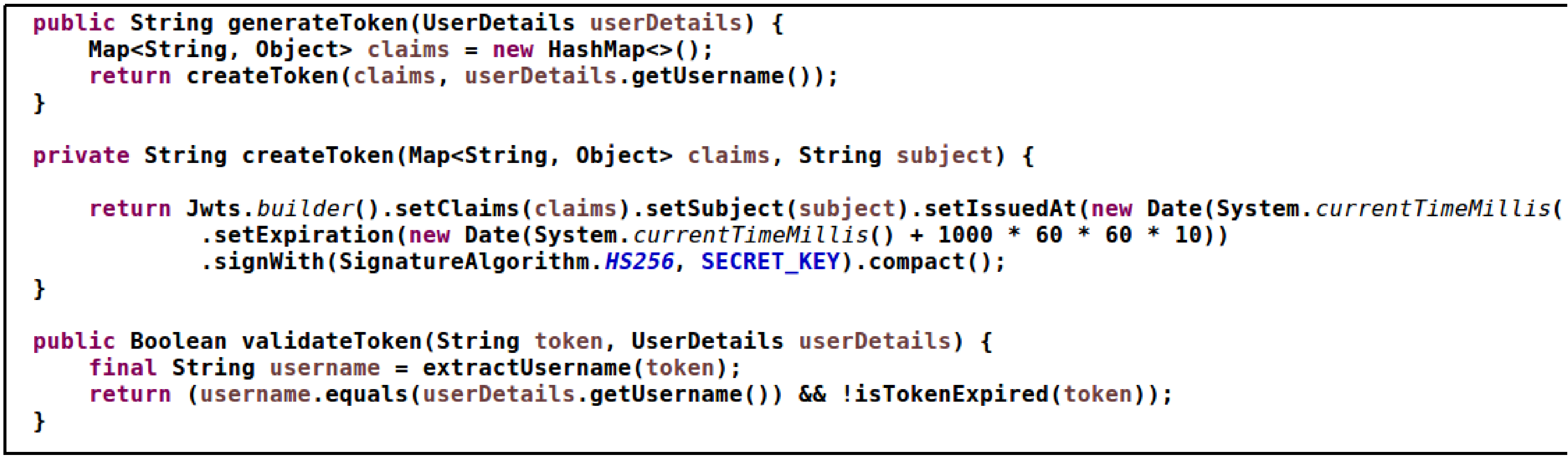
Step2: put jwt dependency and util



```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```



```java
@Service
public class JwtUtil {

    private String SECRET_KEY = "secret";

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }
    private Claims extractAllClaims(String token) {
        return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }
```

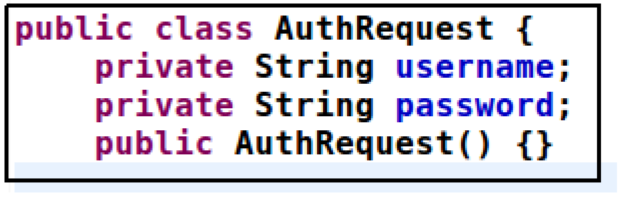


```java
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userDetails.getUsername());
}

private String createToken(Map<String, Object> claims, String subject) {

    return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis(
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY).compact();
}

public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}
```

Step3: create endpoint to accept jwt token

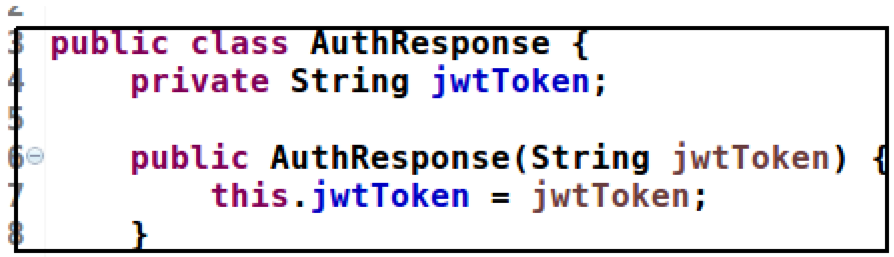
- Accept username and password
- Return jwt token in response

=> create bean to send request



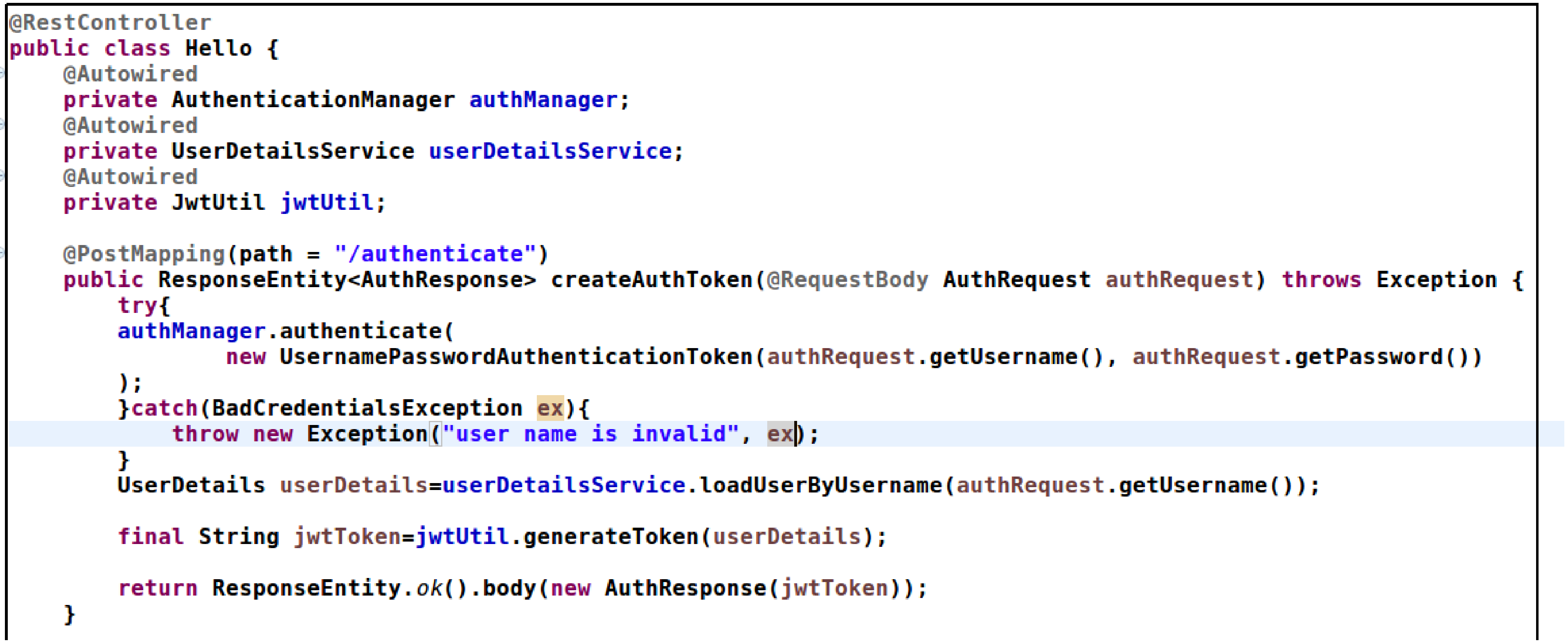
```
public class AuthRequest {
    private String username;
    private String password;
    public AuthRequest() {}
```

=> create bean to get response



```
public class AuthResponse {
    private String jwtToken;

    public AuthResponse(String jwtToken) {
        this.jwtToken = jwtToken;
    }
```

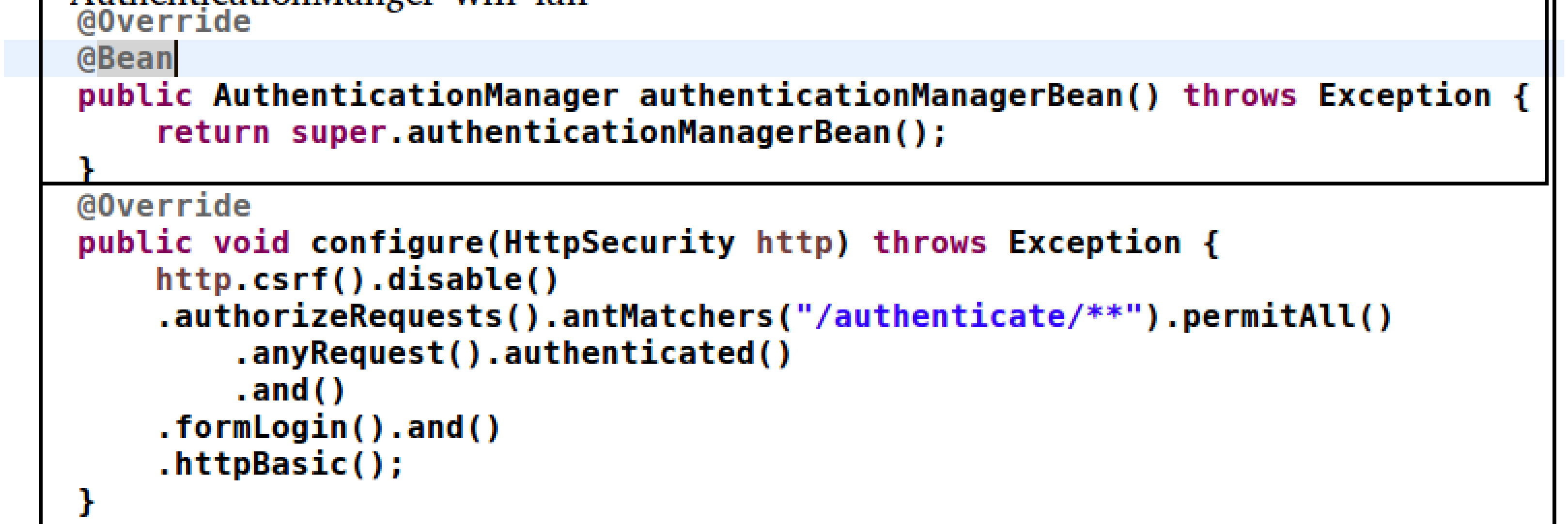
Step3: create endpoint to accept jwt token



```java
@RestController
public class Hello {
    @Autowired
    private AuthenticationManager authManager;
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping(path = "/authenticate")
    public ResponseEntity<AuthResponse> createAuthToken(@RequestBody AuthRequest authRequest) throws Exception {
        try{
        authManager.authenticate(
                new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword())
        );
        }catch(BadCredentialsException ex){
            throw new Exception("user name is invalid", ex);
        }
        UserDetails userDetails=userDetailsService.loadUserByUsername(authRequest.getUsername());

        final String jwtToken=jwtUtil.generateToken(userDetails);

        return ResponseEntity.ok().body(new AuthResponse(jwtToken));
    }
```

Step4: Permit all /authenticate to accept jwt token

Dont forget to Override this method otherwise @Autowire AuthenticationManger will fail

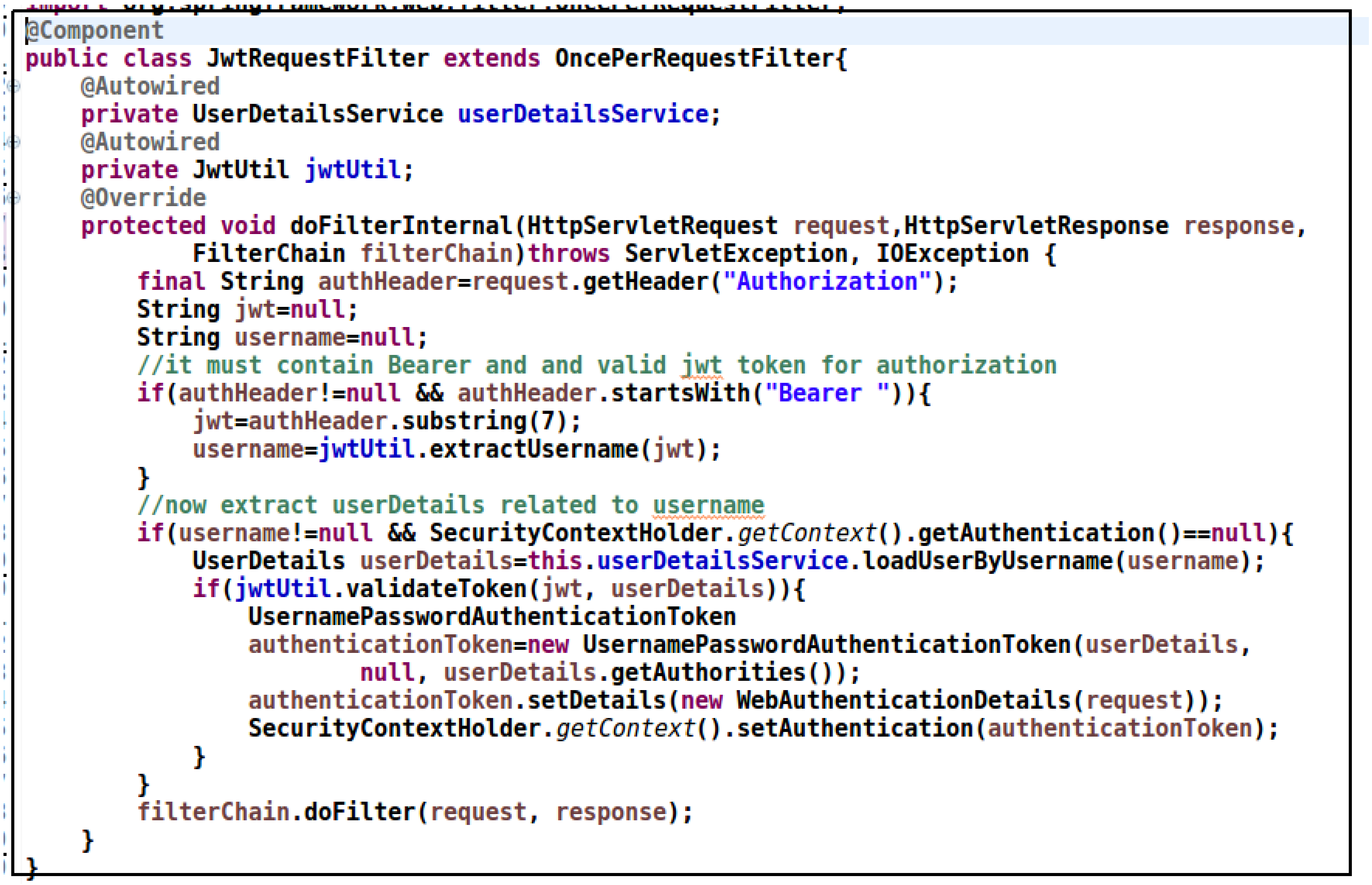


```java
@Override
@Bean
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}
@Override
public void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests().antMatchers("/authenticate/**").permitAll()
        .anyRequest().authenticated()
        .and()
    .formLogin().and()
    .httpBasic();
}
```

Step 5

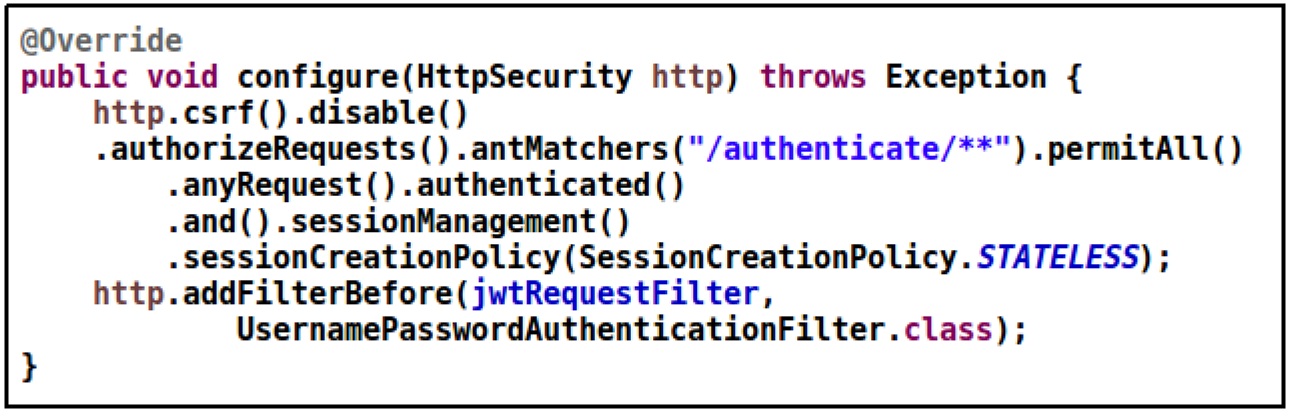
- Intercept all incoming request
  - Extract JWT token for the header
  - Validate and set in execution context

Intercept all incoming request

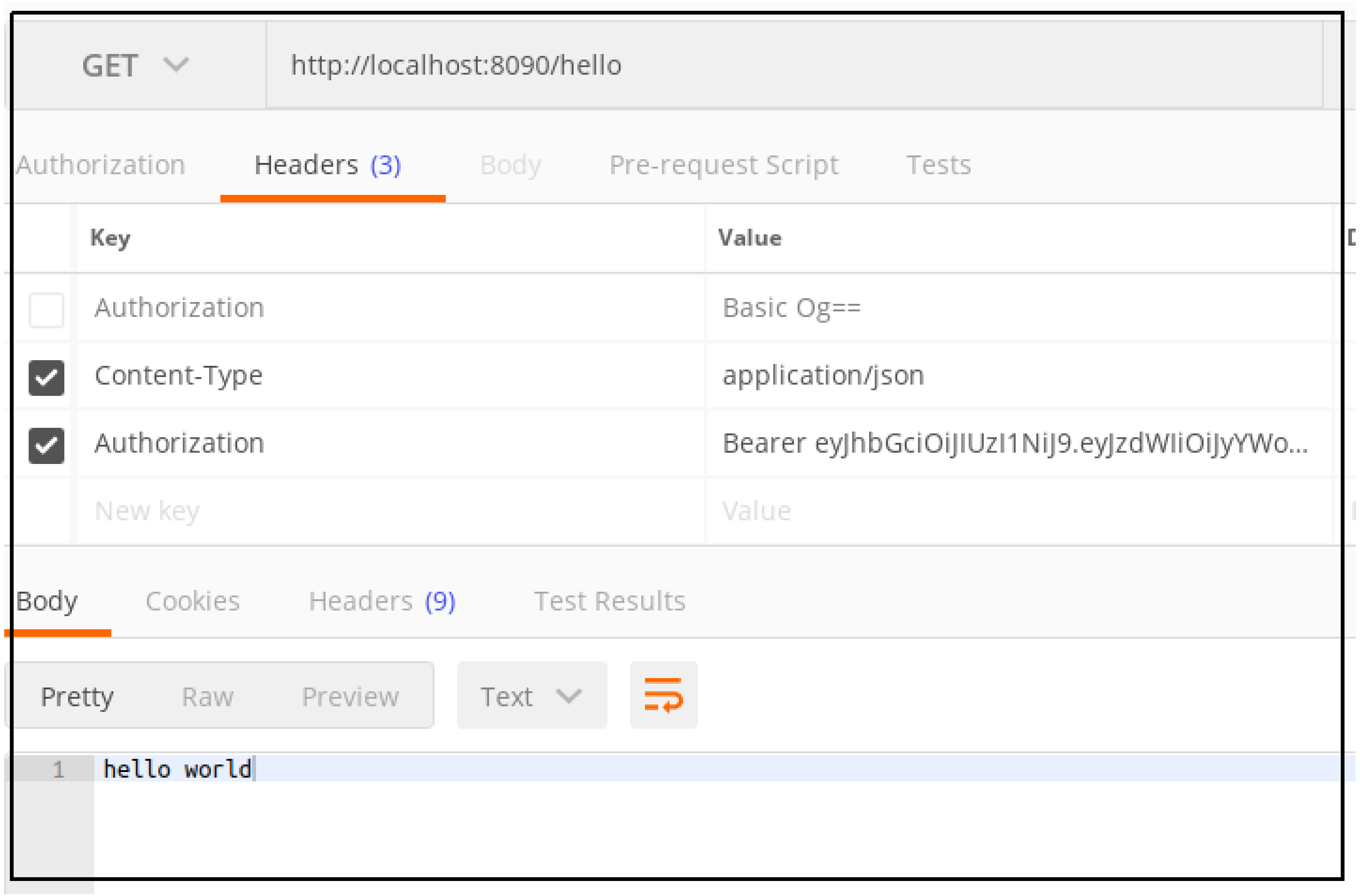


```
@Component
public class JwtRequestFilter extends OncePerRequestFilter{
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private JwtUtil jwtUtil;
    @Override
    protected void doFilterInternal(HttpServletRequest request,HttpServletResponse response,
            FilterChain filterChain)throws ServletException, IOException {
        final String authHeader=request.getHeader("Authorization");
        String jwt=null;
        String username=null;
        //it must contain Bearer and and valid jwt token for authorization
        if(authHeader!=null && authHeader.startsWith("Bearer ")){
            jwt=authHeader.substring(7);
            username=jwtUtil.extractUsername(jwt);
        }
        //now extract userDetails related to username
        if(username!=null && SecurityContextHolder.getContext().getAuthentication()==null){
            UserDetails userDetails=this.userDetailsService.loadUserByUsername(username);
            if(jwtUtil.validateToken(jwt, userDetails)){
                UsernamePasswordAuthenticationToken
                authenticationToken=new UsernamePasswordAuthenticationToken(userDetails,
                        null, userDetails.getAuthorities());
                authenticationToken.setDetails(new WebAuthenticationDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Appying filter to customized security


```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests().antMatchers("/authenticate/**").permitAll()
        .anyRequest().authenticated()
        .and().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtRequestFilter,
            UsernamePasswordAuthenticationFilter.class);
}
```

Testing with jwt token

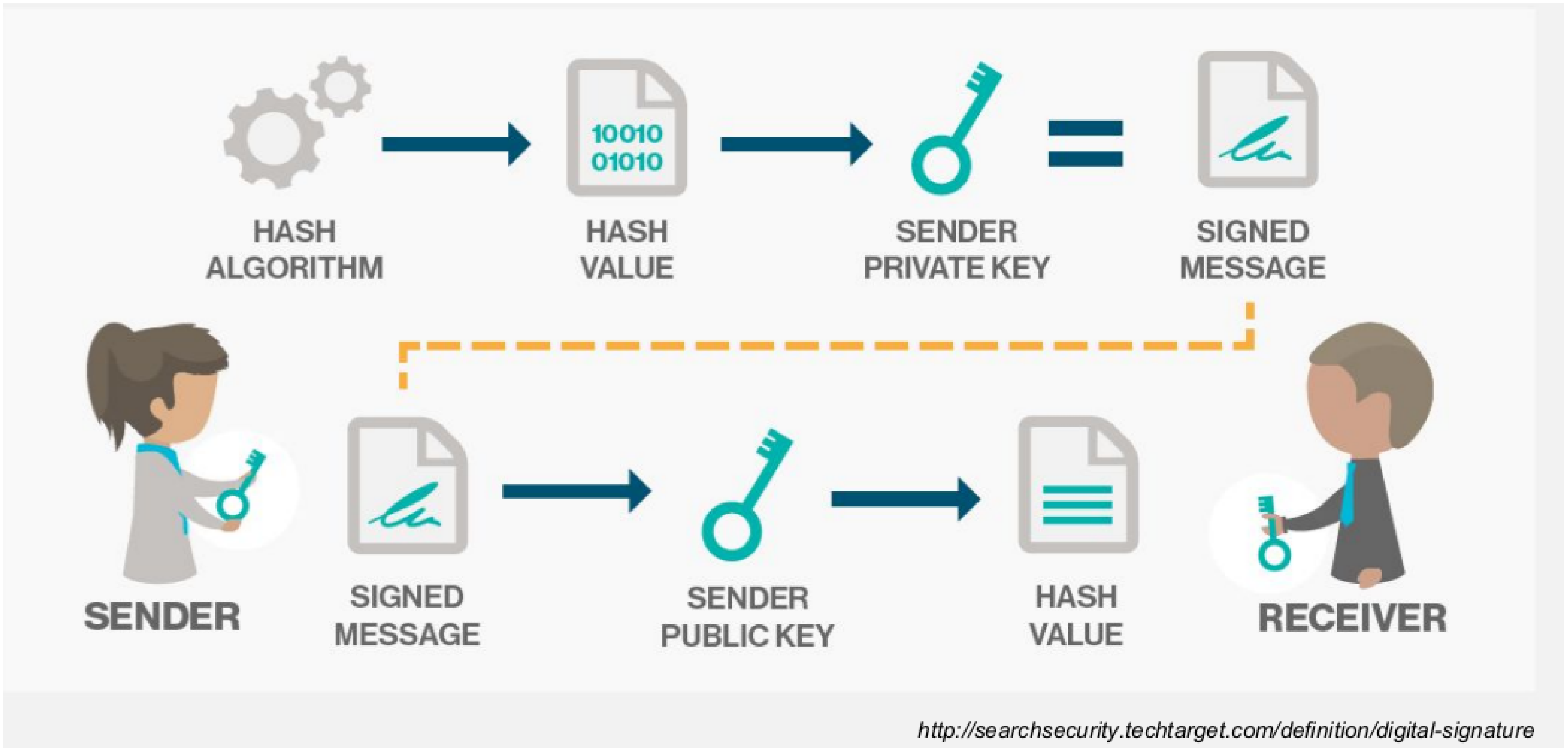


GET http://localhost:8090/hello

	Key	Value
☐	Authorization	Basic Og==
✓	Content-Type	application/json
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJyYWo...
	New key	Value

```
hello world
```

Digital signature process



http://searchsecurity.techtarget.com/definition/digital-signature